TA Session 2

Anthony Gillioz
Institute of Computer Science – University of Bern, Switzerland

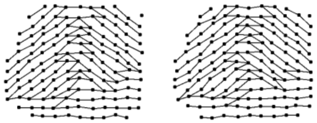
Contact: anthony.gillioz@unibe.ch

Theoretical Tasks

Task 1

1. Consider the graphs illustrated below. Note that in this example each node is labeled with a two-dimensional attribute giving its position in the plane (the graphs represent the ridge structure of two fingerprint images).

   Elaborate on the problems of exact graph matching paradigms in this particular setting (or vice versa on the benefits of error-tolerant graph matching).

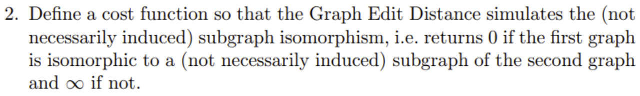


a) b)

- Describe the problem using the graph-matching paradigm.
- Which paradigm would you choose for this application: Exact graph matching or error-tolerant graph matching? Justify your decision in detail!
- Hint: Since both graphs depict the same fingerprint image, their dissimilarity score should be close to zero (which will not be the case for any exact GM algorithm).

Theoretical Tasks

Task 2



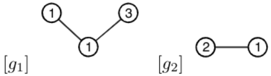
2. Define a cost function so that the Graph Edit Distance simulates the (not necessarily induced) subgraph isomorphism, i.e. returns 0 if the first graph is isomorphic to a (not necessarily induced) subgraph of the second graph and ∞ if not.

- Define the cost function for node insertion, deletion, and substitution.
- Define the cost function for edge insertion, deletion, and substitutions.

Theoretical Tasks

Task 3

3. Expand the search tree for computing the edit distance between the following two graphs.



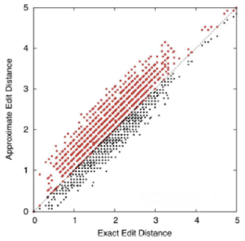
The nodes are labeled with integers, while the edges are unlabeled. You can use unit cost for deletions and insertions of both nodes and edges. Edge substitutions are free of cost, while the cost for substituting a node $u \in V_1$ with a node $v \in V_2$ is defined via $c(u \rightarrow v) = |\mu_1(u) - \mu_2(v)|$. Set $h(\lambda) = 0$ for all edit paths, i.e. use no heuristic information.

- Draw the search tree based on the exact graph edit distance algorithm (Alg. 3 in the lecture notes)
- Refer to Fig. 3.5 in the lecture notes for an example of what the drawing should look like.

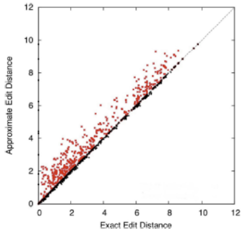
Theoretical Tasks

Task 4

4. Consider the scatter plots in the figure below. They show the exact (x-axis) vs. the approximate (y-axis) graph edit distance computed with BP-GED resulting in d_ψ (red points) and d'_ψ (black points) on two different graph data sets. Elaborate on these figures (What can be observed?)



(a) AIDS



(b) FP

- Discuss the insights that can be drawn from analyzing the plots.
- Compare and contrast the distances obtained using the GED algorithm with those obtained using the BP-GED algorithm.

Implementation Tasks

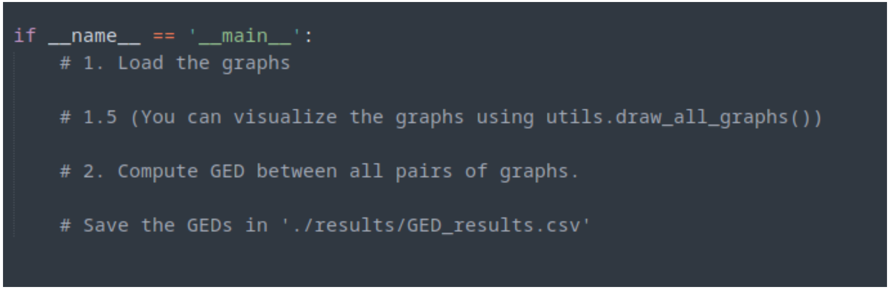
In this implementation task, the goal is to implement (lower- or upper-bound) BP-GED presented in the lecture notes (Chap. 4).

Remarks:

- The entire code must be contained within the file `PR_lecture/Exercise_2/ex2.py`
- You are allowed to modify the code as much as you want, including changing function signatures, creating new functions or classes, and so on.

Implementation Tasks

Idea of code structure

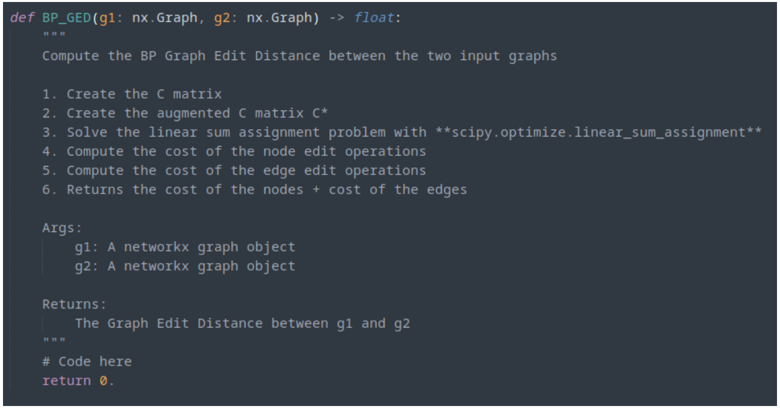


```
if __name__ == '__main__':
    # 1. Load the graphs

    # 1.5 (You can visualize the graphs using utils.draw_all_graphs())

    # 2. Compute GED between all pairs of graphs.

    # Save the GEDs in './results/GED_results.csv'
```

UNIVERSITÄT BERN

Implementation Tasks

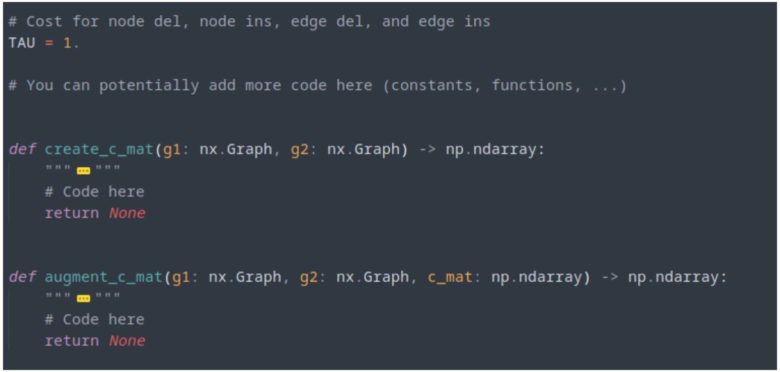
Idea of code structure



```python
def BP_GED(g1: nx.Graph, g2: nx.Graph) -> float:
    """
    Compute the BP Graph Edit Distance between the two input graphs

    1. Create the C matrix
    2. Create the augmented C matrix C*
    3. Solve the linear sum assignment problem with **scipy.optimize.linear_sum_assignment**
    4. Compute the cost of the node edit operations
    5. Compute the cost of the edge edit operations
    6. Returns the cost of the nodes + cost of the edges

    Args:
        g1: A networkx graph object
        g2: A networkx graph object

    Returns:
        The Graph Edit Distance between g1 and g2
    """
    # Code here
    return 0.
```

Implementation Tasks

Idea of code structure



```
# Cost for node del, node ins, edge del, and edge ins
TAU = 1.

# You can potentially add more code here (constants, functions, ...)


def create_c_mat(g1: nx.Graph, g2: nx.Graph) -> np.ndarray:
    """...."""
    # Code here
    return None


def augment_c_mat(g1: nx.Graph, g2: nx.Graph, c_mat: np.ndarray) -> np.ndarray:
    """...."""
    # Code here
    return None
```

Implementation Tasks

Remarks

You have to define the cost function for node and edge substitution (as defined in the exercise series 2) (i.e., node sub: $c(u \rightarrow u') = |\mu_1(u) - \mu_2(u')|$ and edge sub $c((u,v) \rightarrow (u',v')) = 0$).

- The node substitution function usually takes in two node labels as inputs and calculates the absolute difference between them.
- The edge substitution cost being zero simplifies the construction of the $\mathbf{C}^*$ matrix. Essentially, this means that when adding the structural relationships to $\mathbf{C}$, we only need to consider the difference in node degrees between the nodes being compared.